JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 13 (2019) 339-344
Wysłane: 2019-09-07
Przyjęte: 2019-09-17

Analiza możliwości obrony przed atakami SQL Injection

Chrystian Byzdra*, Grzegorz Kozieł
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule scharakteryzowano różne metody ochrony bazy danych oraz typy ataków SQL Injection. Są to wyjątkowo niebezpieczne ataki, ponieważ zagrażają poufności wrażliwych danych. W celu szczegółowej analizy metod ochrony i sposobów ataków zostały wykonane symulacje ataków i obrony w językach : C#, PHP, Java. Na podstawie wyników symulacji dla poszczególnych języków porównano skuteczność oraz wydajność metod ochrony bazy danych.

Słowa kluczowe: wstrzyknięcie kodu SQL; ochrona; sprawdzanie danych wejściowych

*Autor do korespondencji.

Adres e-mail: cbyzdra@gmail.com

Analysis of the defending possibilities against SQL Injection attacks

Chrystian Byzdra*, Grzegorz Kozieł
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article describes various protection methods of database and types of SQL Injection attacks. These are extremely dangerous attacks because they threaten the confidentiality of sensitive data. In order to analyze in detail protection methods and methods of attacks, simulations of attacks and defence were performed in the following languages: C #, PHP, Java. Based on the simulation results for particular languages, the effectiveness and efficiency of database protection methods were compared.

Keywords: SQL injection; prevention; input validation

*Corresponding author.

E-mail address: cbyzdra@gmail.com

1. Wstęp

Narzędziem umożliwiającym komunikację z bazą danych jest język SQL. Poprzez formularz w aplikacji atakujący może dodać złośliwy kod do zapytania SQL i wykonać na niej dowolne operacje. Takie działanie jest nazywanie atakiem SQL Injection. Używając tej metody atakujący jest w stanie wykorzystać lub modyfikować wrażliwe informacje. Niewystarczająca walidacja danych wejściowych umożliwia ten atak i zagraża polityce bezpieczeństwa przechowywanych danych.

Ataki SQLIA (z ang. *SQL injection attack*) są obecnie dość często wykorzystywane do pozyskiwania poufnych informacji o klientach i o poszczególnych zamówieniach z baz danych typu „*e- commerce*", także do obchodzenia mechanizmów bezpieczeństwa [1]. Typowe komunikaty o błędach SQL mogą niekiedy ułatwić atak na bazę danych, ponieważ w przypadku braku wiedzy na temat zapytania SQL lub tabel, technika wymuszania wyjątków okazuje się być skuteczna w praktyce [1]. Udoskonalanie technik programowania jest jednym ze sposobów zapobiegania podobnym atakom. Można wykorzystać do tego m. in. unikanie poszczególnych pojedynczych cudzysłowów, wprowadzić limit długości znaków wejściowych, a także uzupełniać komunikaty o różnych wyjątkach.

2. Ataki SQL Injection

Ataki Sql Injection pojawiły się wraz z pojawieniem bazy danych opartych o język SQL i aplikacji podłączonych do Internetu. Są jednym z bardziej destrukcyjnych podatności prowadzą do ekspozycji wrażliwych danych przechowywanych w aplikacjach. Jest to mocno zaakcentowane we wstępie książki J. Clarke'a [1]. Dla podkreślenia swojej tezy nawiązuje do artykułu, który został opublikowany w 1998 roku. Artykuł opisywał wyżej wymieniony atak na popularną stronę „PacketStorm". W dalszej części książki przedstawia różne sposoby ataków w celu uświadomienia czytelnika o możliwych zagrożeniach.

Temat ataków jest również często poruszany w artykułach naukowych. W swoim artykule Sadeghian [4] uświadamia czytelnika, że ten atak ma najwyższą pozycję w rankingu w podatnościach stron internetowych. Proponuje również podział tych ataków na trzy kategorie [4]:

- w paśmie - dane są pobierane tym samym kanałem, którym przeprowadzany jest atak,
- poza pasmem - pobrane dane są odsyłane do atakującego innym kanałem (np. email),
- inferencyjne – znane również jako ślepe wstrzyknięcie z ang. „*blind injection"*).

Ataki mogą zostać podzielone również pod względem celu atakującego [6]:

- tautologie,
- nielegalne lub logicznie niepoprawne zapytania,
- zapytania z użyciem słowa Union,
- zapytania Piggy-Backed,
- procedury składowane,
- wnioskowanie,
- ślepe wstrzyknięcie,
- ataki czasowe

3. Obrona przed atakiem SQL Injection

W swoje książce Clarke [1] opisuje metody ochrony przed atakami na poziomie warstwy kodu oraz określa następujące metody ochrony przed atakami:

- wyrażenia parametryzowane,
- sprawdzanie poprawności danych wejściowych,
- znak wyjścia,
- procedury składowane,
- warstwy abstrakcji

W literaturze zaproponowane są również inne podejścia do obrony przed SQLIA. Lambert [7] proponuje podzielenie zapytania spodziewanego i otrzymanego względem znaków takich jak „' ” lub „ ” oraz porównanie długości w otrzymanych tablicach. Jeżeli długości otrzymanych tablic są różne, to mechanizm stwierdza SQLIA.

Kolejny ciekawy wątek został poruszony przez Kar i Panigrahi [8]. Twórcy artykułu demonstrują technikę polegającą na transformacji zapytania z formy strukturalnej na formę parametryzowaną. W celu utworzenia pełnego zapytania, autorzy utworzyli schemat transformacji. Schemat (w postaci tabeli) został utworzony tak, aby obsłużyć dużą ilość możliwych zapytań. W celu przechowywania transformowanych zapytań i ich szybkiego wyszukiwania, postanowiono skorzystać z funkcji skrótu MD5 (z ang. *Message-Digest algorithm 5*).

Inne podejście zostało przedstawione przez Amutha Prabakar [9], czyli identyfikowanie podejrzanego kodu poprzez wyszukiwanie wzorca we wprowadzonych danych przez użytkownika. Jest wiele różnych podejść do rozpoznawania wzorca w tekście używających automatów skończonych. Algorytm użyty do wyszukiwania wzorca w tekście to Aho-Corasick.

Do ochrony przed SQLIA można użyć wcześniej wspomnianych procedur składowanych [10] (z ang. *„Stored Procedures”*). Wei przedstawia technikę obrony opartą o procedurę składowaną. Ta procedura zawiera parser dla każdego zapytania SQL wprowadzanego przez użytkownika [10]. Jeżeli zapytanie zmieni strukturę oryginalnej instrukcji to atak zostanie zidentyfikowany.

W niniejszej pracy przebadano wybrane metody obrony przed atakami SQL Injection. Należą do nich:

- wyrażenia parametryzowane,
- procedury składowane,
- funkcja skrótu – MD5,
- znak wyjścia,
- tokenizacja zapytania

4. Symulacje ataków i metod obrony

Symulacje ataków przeprowadzono z uwzględnieniem języka w jakim został zaimplementowany kod aplikacji uzyskującej dostęp do bazy danych oraz specyficznych dla nich bibliotek i wzorców programistycznych. Typ ataku wykorzystany do symulacji został oparty o tautologie.

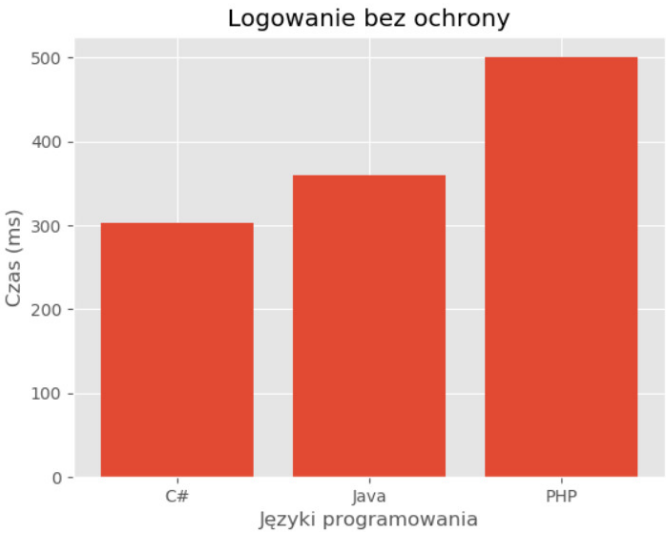
4.1. Logowanie bez ochrony

Logowanie bez ochrony zostało zrealizowane niezależnie w każdym z badanych języków programowania poprzez pobranie danych z formularza i przesłanie ich bezpośrednio do serwera bazodanowego w postaci nieparametryzowanego zapytania SQL. Przykład realizacji tej funkcji w języku C# przedstawiono na przykładzie 1.

Przykład 1. Logowanie niezabezpieczone w języku C#

```
sqlCommand.Connection = connection;
sqlCommand.CommandText = @"SELECT * FROM Users
WHERE UserLogin ='"
+ loginCredentials.email + "' AND UserPass='"
+ loginCredentials.password + "';";
SqlDataReader reader;
reader = sqlCommand.ExecuteReader();
if (reader.Read())
{ ret = true;
}
```

Badanie polegało na zmierzeniu czasu logowania we wszystkich trzech językach. Uzyskane wyniki przedstawiono na Rysunku 1.



Rys. 1. Porównanie czasów logowania bez ochrony dla 1000 prób

4.2. Wyrażenia parametryzowane

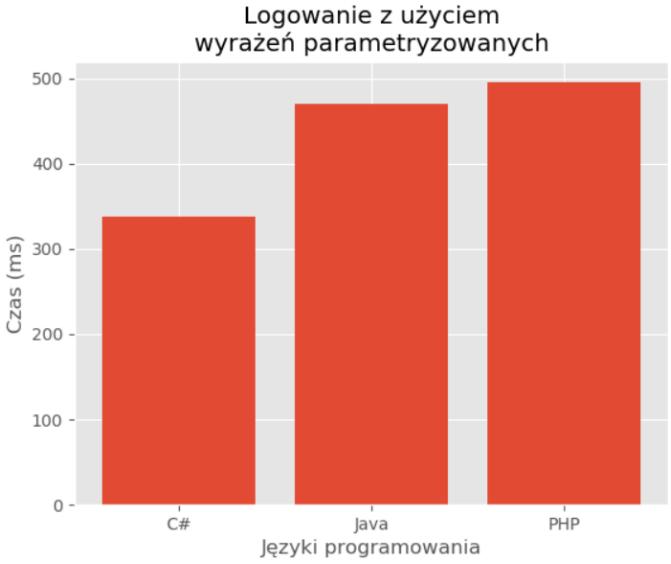
Nowoczesne języki programowania umożliwiają dostarczanie parametrów do zapytania SQL poprzez użycie

symboli zastępczych lub zmiennych wiążących, zamiast pracować bezpośrednio z danymi wejściowymi użytkownika. Powszechnie znane jako wyrażenia parametryzowane. Implementacja wyrażeń parametryzowanych w języku Java została przedstawiona na Przykładzie 2.

Przykład 2. Realizacja logowania z użyciem wyrażeń parametryzowanych w Java

```
String queryString = "SELECT * FROM Users WHERE
UserLogin =? AND UserPass=?;";
PreparedStatement statement =
con.prepareStatement(queryString);
statement.setString(1, _loginCredentials.email);
statement.setString(2, _loginCredentials.password);
ResultSet rs = statement.executeQuery();
if (rs.next()) {
ret = true;
}
```

Na Rysunku 2 zostały pokazane wyniki badań. Przedstawiono na nim wykresy czasu logowania używając procedur składowanych dla 1000 prób.



Rys. 2. Porównanie czasów logowania z użyciem wyrażeń parametryzowanych dla 1000 prób

4.3. Procedury składowane

Procedury składowane są ważną częścią współczesnej relacyjnej bazy danych. Dodają one dodatkową warstwę abstrakcji do systemu oprogramowania. Procedury składowane mogą być bardzo przydatne w ograniczaniu potencjalnej podatności na wstrzyknięcie SQL. Sposób wywołania procedury składowanej w PHP został przedstawiony na Przykładzie 3.

Przykład 3. Realizacja logowania z użyciem procedur składowanych w PHP

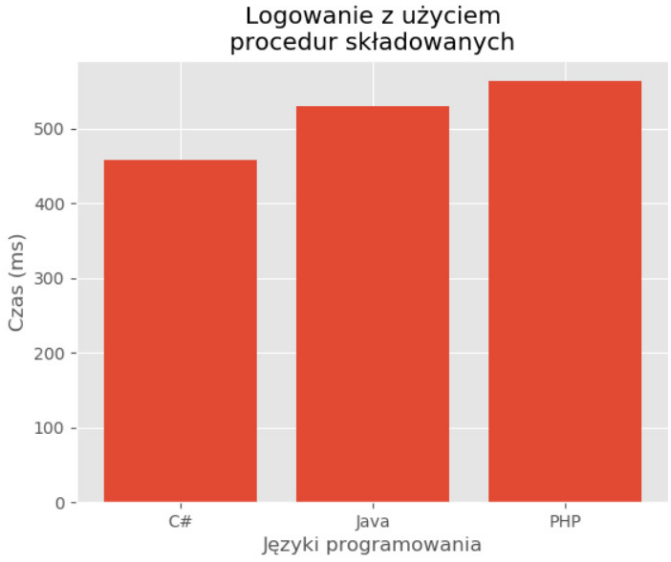
```
$stmt = $conn->prepare('{CALL loginCheck(?,?)}');
    $stmt->bindParam(1, $email, PDO::PARAM_STR |
pdo::PARAM_INPUT_OUTPUT, 50);
$stmt->bindParam(2, $pass, PDO::PARAM_STR |
pdo::PARAM_INPUT_OUTPUT, 50);
$result =  $stmt->execute();
$row = $stmt->fetch( PDO::FETCH_ASSOC );
if($row)
{
return true;
}
```

W Przykładzie 3 jest zawarta instrukcja wywołująca procedurę składowaną, która jest widoczna na Przykładzie 4.

Przykład 4. Procedura składowana **LoginCheck**

```
CREATE PROCEDURE [dbo].[LoginCheck]
@login varchar(50) output,
@pwd varchar(50) output
AS
select * from Users
    where  UserLogin = @login and UserPass = @pwd
return
```

Rysunek 3 przedstawia wyniki zrealizowanych symulacji dla logowania z użyciem procedur składowanych. Na podstawie wykresów można wywnioskować, że dłużej wykonuje się logowanie z użyciem procedur składowanych niż przy pomocy wyrażeń parametryzowanych.



Rys. 3. Porównanie czasów logowania z użyciem procedur składowanych dla 1000 prób

4.4. Funkcja skrótu MD5

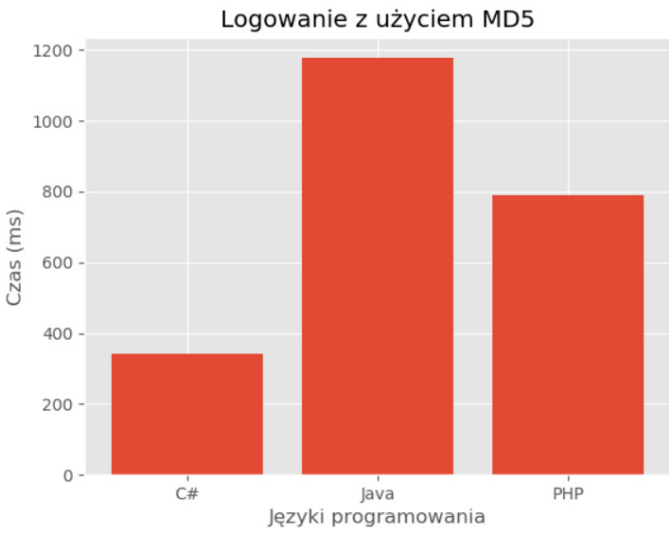
Wiele zapytań można zredukować do tej samej struktury [8]. Forma strukturalna wstrzykniętego zapytania różni się zasadniczo od poprawnej kwerendy, więc można ją łatwo określić jako SQLIA. Redukując zapytania do tej samej formy strukturalnej minimalizowany jest rozmiar przestrzeni wyszukiwania dla dopasowania w czasie wykonywania. W celu przyspieszenia wyszukiwania formę strukturalną zapytania przedstawia się w postaci wyniku funkcji skrótu MD5. Na Przykładzie 5 została przedstawiona zmodyfikowana wersja transformacji zapytania i kodowania, która ogranicza się do obliczenia MD5 dla danych

wejściowych i porównaniu wyniku z obliczonymi wartościami przechowywanymi w bazie danych.

Przykład 5. Realizacja logowania z użyciem MD5 w PHP

```
$hashedLogin = md5($email);
$hashedPass = md5($pass);
$tsql = "select * from Users as u where
CONVERT(VARCHAR(32), HashBytes('MD5', u.UserLogin), 2)
='".$hashedLogin."'and CONVERT(VARCHAR(32),
HashBytes('MD5', u.UserPass), 2) ='" .$hashedPass."';";
$getResults = $conn->prepare($tsql);
$getResults->execute();
$results = $getResults->fetchAll(PDO::FETCH_BOTH);
if($results)
{
return true;
}
```

Na Rysunku 4 przedstawione zostały wyniki czasów logowania z wykorzystaniem funkcji MD5 w trzech językach. W tej symulacji najdłuższy czas wykonania miała metoda napisane w języku Java. Na czas realizacji funkcji ma wpływ szybkość obliczenia wyniku funkcji skrótu.



Rys. 4. Porównanie czasów logowania z użyciem MD5 dla 1000 prób

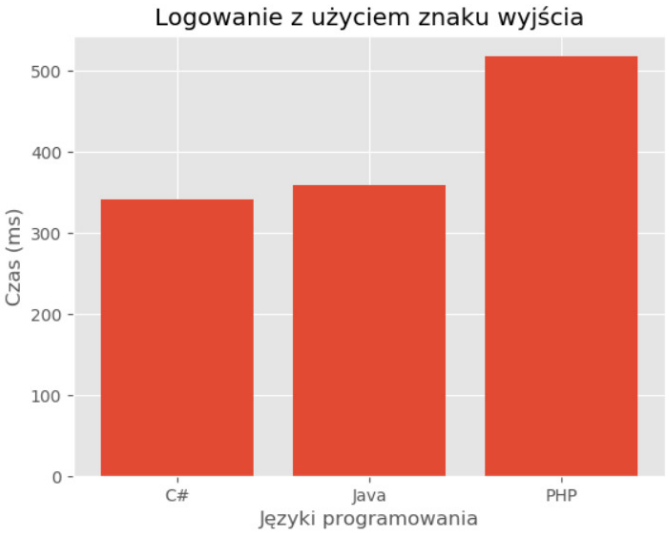
4.5. Znak wyjścia

Oprócz sprawdzania poprawności danych wejściowych odbieranych przez aplikację często konieczne jest również zakodowanie tego, co jest przekazywane między różnymi modułami lub częściami aplikacji. W SQL jest używany pojedynczy cudzysłów jako koniec dla wprowadzonej treści przez użytkownika. Wprowadzenie napisu posiadającego pojedynczy cudzysłów jest problematyczne. Rozwiązaniem tego problemu i jednocześnie zabezpieczeniem przed SQLIA jest zakodowanie napisu poprzez dodaniu znaku wyjścia. Na Przykładzie 6 została zaprezentowana przykładowa metoda napisana w języku C#, która zamienia pojedynczy cudzysłów na podwójny.

Przykład 6. Logowanie z użyciem znaku wyjścia w C#

```
string preparedEmail, preparedPass;
preparedEmail = loginCredentials.email.Replace("'", "''");
preparedPass = loginCredentials.password.Replace("'", "''");
SqlCommand sqlCommand = new SqlCommand();
sqlCommand.Connection = connection;
sqlCommand.CommandText = @"SELECT * FROM Users
WHERE UserLogin ='"
+ preparedEmail + "' AND UserPass='" + preparedPass +
"';";
SqlDataReader reader;
reader = sqlCommand.ExecuteReader();
if (reader.Read())
{
ret = true;
}
```

Na Rysunku 5 zostały przedstawione wyniki symulacji logowania z użyciem znaku wyjścia dla wybranych języków programowania. Wskazane czasy na wykresie są bardzo zbliżone do czasów logowania bez metody ochrony.



Rys. 5. Porównanie czasów logowania z użyciem znaku wyjścia dla 1000 prób

4.6. Tokenizacja zapytania

Metoda polegająca na dzieleniu wprowadzonego zapytania względem ustalonych tokenów, Najczęściej wybieranymi tokenami są pojedynczy cudzysłów oraz spacja. Po podzieleniu napisu i wprowadzeniu wartości do tablicy lub listy, można porównywać właściwości tych struktur danych z właściwościami oczekiwanymi. Na Przykładzie 7 została przedstawiona realizacja tej metody w języki Java.

Przykład 7. Realizacja logowania z użyciem tokenizacji zapytania w języku Java

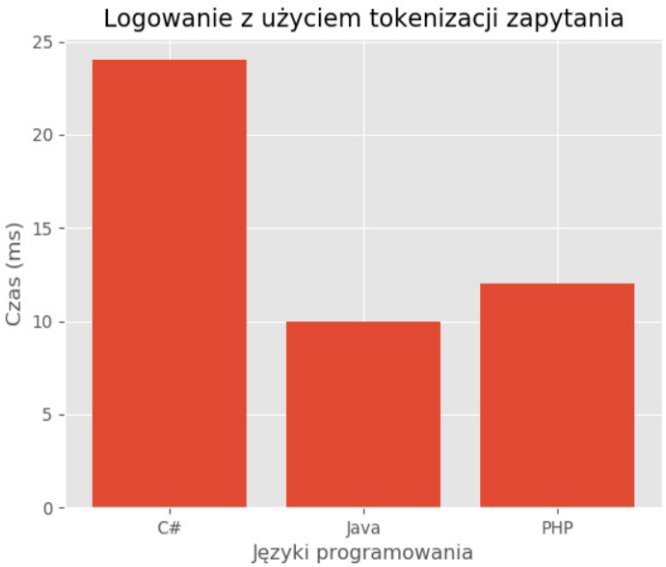
```
String queryToCmp = "SELECT * FROM Users WHERE
UserLogin ='' AND UserPass='';";
String insertedQuery = "SELECT * FROM Users WHERE
UserLogin ='"
+ _loginCredentials.email + "' AND UserPass='"
+ _loginCredentials.password + "';";
if (LoginCredentials.validateByQueryToken(insertedQuery,
queryToCmp))
```

Sprawdzenie poprawności zapytania jest realizowane przed nawiązaniem połączenia z bazą danych. Implementacja metody walidującej zapytanie została pokazana na Przykładzie 8.

Przykład 8. Metoda sprawdzająca wprowadzone dane

```
tokensToCheck.add(' ');
tokensToCheck.add(';');
for (Character item : tokensToCheck) {
tokensInsQuery = _insertedQuery.split(item.toString());
tokensQueryToCmp = _queryToCmp.split(item.toString());
if (tokensInsQuery.length != tokensQueryToCmp.length) {
ret = false;
break;
```

Na Rysunku 6 zostały zaprezentowane wyniki symulacji. Najniższy czas wykonania miała funkcja zaimplementowana w języku Java. Czas wykonania metody jest związany z bardzo szybkim dostępem do struktur danych w tym języku. Na wynik nie miała wpływu implementacja sterownika, ponieważ system w przypadku wykrycia SQLIA nie uruchamiał procesu połączenia z bazą.

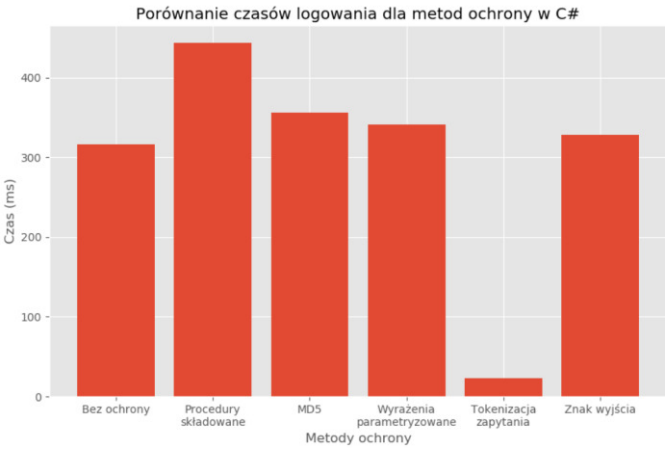


Rys. 6. Porównanie czasów logowania z użyciem tokenizacji zapytania dla 1000 prób

5. Wnioski

Celem pracy było porównanie metod ochrony przed atakiem SQL Injection. Wykonano programy, które symulowały obronę przed wstrzykniętym kodem SQL. Porównanie czasów wykonania symulacji w języku C# dla 1000 prób zostało zaprezentowane na Rysunku 7.

Metody obrony oparte na analizie wprowadzonego tekstu (np. tokenizacja zapytania) są zdecydowanie najszybsze, co wynika z faktu, że w przypadku wykrycia złośliwego kodu SQL nie zostaje nawiązywane połączenie z bazą danych. Jednak tokenizacja zapytania nie jest w pełni bezpieczna. W przypadku wprowadzenia danych w zakodowanej postaci (np. heksadecymalnej) funkcja nie wykryje ataku. Dosyć szybkim sposobem ochrony okazała się metoda dodająca znak wyjścia do zapytania. Jest to bezpieczne rozwiązanie, aczkolwiek należy pamiętać, że dla niektórych danych takie zapytania mogą być mało czytelne. Najmniej wydajną metodę stanowiła oparta na funkcji skrótu MD5, jednakże jest ona jednocześnie bardzo bezpieczna dla wybierania informacji, ponieważ wszystkie wprowadzone dane zostają kodowane. W rozbudowanych zapytaniach w aplikacjach biznesowych, obliczanie funkcji skrótu dla każdej wprowadzonej danej może być problematyczne.



Rys. 7. Porównanie czasów logowania dla metod ochrony w C#

Rozwiązanie zabezpieczające przed wstrzyknięciem złośliwego kodu to także procedury składowane. W przypadku poprawnego zaimplementowania są bezpieczne, szybkie oraz mogą zostać ponownie wykorzystane w innych miejscach w aplikacji. Jako wadę takiego rozwiązania należy wskazać fakt, że w przypadku zmiany bazy danych, wszystkie procedury musiałyby zostać na nowo zaimplementowane.

Następną metodą, która posiada wiele zalet jest oparta na wyrażeniach parametryzowanych. Działa ona szybko, implementowana zostaje w kodzie niezależnie od bazy danych i zachowuje bezpieczeństwo danych. Wbudowane mechanizmy, napisane w danym języku zapewniają wprowadzenie informacji do zapytania tylko w postaci napisu.

Literatura

[1] Clarke J.: SQL Injection Attacks and Defense, Syngress Publishing, 2009

[2] Somesh J., Christodorescu M., Wang C., Maughan D., Song D.: Malware Detection, Springer, 2006

[3] Snyder C., Southwell M.: Pro PHP Security, Apress, 2005

[4] Sadeghian A., Zamani M., Ibrahim S.: SQL Injection is Still Alive:A Study on SQL Injection Signature Evasion Techniques, IEEE, 2013

[5] Heydari M.Z.: Comparision of SQL Injection Detection and Prevention Techniques, ICETC, 2010

[6] Halfond W.G.J., Viegas J., Orso A.: A Classification of SQL Injection Attacks and Countermeasures, IEEE, 2006

[7] Lambert N., Song Lin K.: Use of Query Tokenization to detect and prevent SQL Injection Attacks, IEEE, 2010

[8] Kar D., Panigrahi S.: Prevention of SQL Injection Attack Using Query Transformation and Hashing, IEEE, 2012

[9] Amutha Prabakar M., KarthiKeyan M., Marimuthu K.: An efficient technique for preventing SQL Injection attack using pattern matching algorithm, IEEE, 2013

[10] Wei K., Muthuprasanna M., Kothari S.: Preventing SQL Injection Attacks in Stored Procedures, IEEE, 2006

[11] Specyfikacja języka C# http://docs.microsoft.com/pl-pl/dotnet/csharp/language-reference/language-specification/introduction [20.05.2019]

[12] Podstawy programowania w języku Java, https://docs.oracle.com/javase/tutorial/java/index.html [13.05.2019]

[13] Dokumentacja techniczna języka PHP, https://www.php.net/manual/en/ [11.04.2019]

[14] Opis standardów i składni języka SQL, http://bazy.rzeszow.pl/klasy/klasa3bazy/sql.pdf [15.05.2019]

[15] Wykład z języka SQL przedstawiający podstawowe funkcje, https://www.mechanikryki.pl/renata/pliki_pdf/SQL.pdf [15.05.2019]